Calvin Zikakis

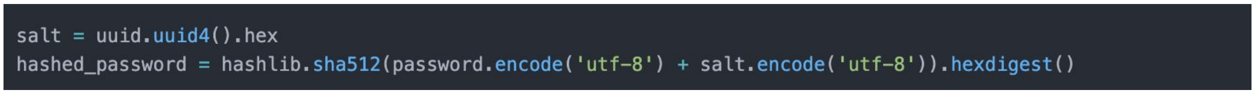
Maura Kieft

Borui Yu

Intro to Cybersecurity - Project 2

Password Hashing

For hashing the password in the file, we imported hashlib and used a SHA-2 hashing algorithm which is a form of secure hashing algorithm developed by the United States National Security Agency. We decided on this hashing algorithm as it provides great security against collision attacks while still having good performance. For salting we generated a random uuid.hex with the help of the uuid library. This provides us with a 32-character hexadecimal string that we then append to the password before hashing the result. This greatly increases our security against precomputed dictionary attacks.



```
salt = uuid.uuid4().hex
hashed_password = hashlib.sha512(password.encode('utf-8') + salt.encode('utf-8')).hexdigest()
```

Public and Private Key Usage

Our private key and public key pair were generated using openssl with a bit size of 1024. The following command allowed us to generate the private key with RSA encryption:

```
openssl genrsa -out rsa.private 1024
```

Next, to generate our public key. We run the following command:

```
openssl rsa -in rsa.private -out rsa.public -pubout -outform PEM
```

This leaves us with a private / public key pair using RSA encryption. Our keys were stored in a folder titled “TheKeys.” We imported AES from Crypto.Cipher to encrypt the messages with

Cipher-Block Chaining. This CIpher-Block Chaining forces us to use an initialization vector, so we randomly picked one. This ciphertext method depends on all previous plaintext blocks and the current as well. AES allows us to encrypt with a mode of operation which provides security for the messages between the server and client.

Symmetric Encryption

In order to manage symmetric encryption, we used the Advanced Encryption Standard (AES) which is an iterative block cipher. Encrypting converts the data into ciphertext and decrypting converts the data back into plaintext. This encryption mode was chosen because its operational time is fast and compact on a variety of platforms, the most secure, and it can resist most, if not all, attacks. For this encryption mode we did have to trade off a higher memory in comparison to RC5 and energy consumption in comparison to other block cipher encryption algorithms.

Eavesdroppers

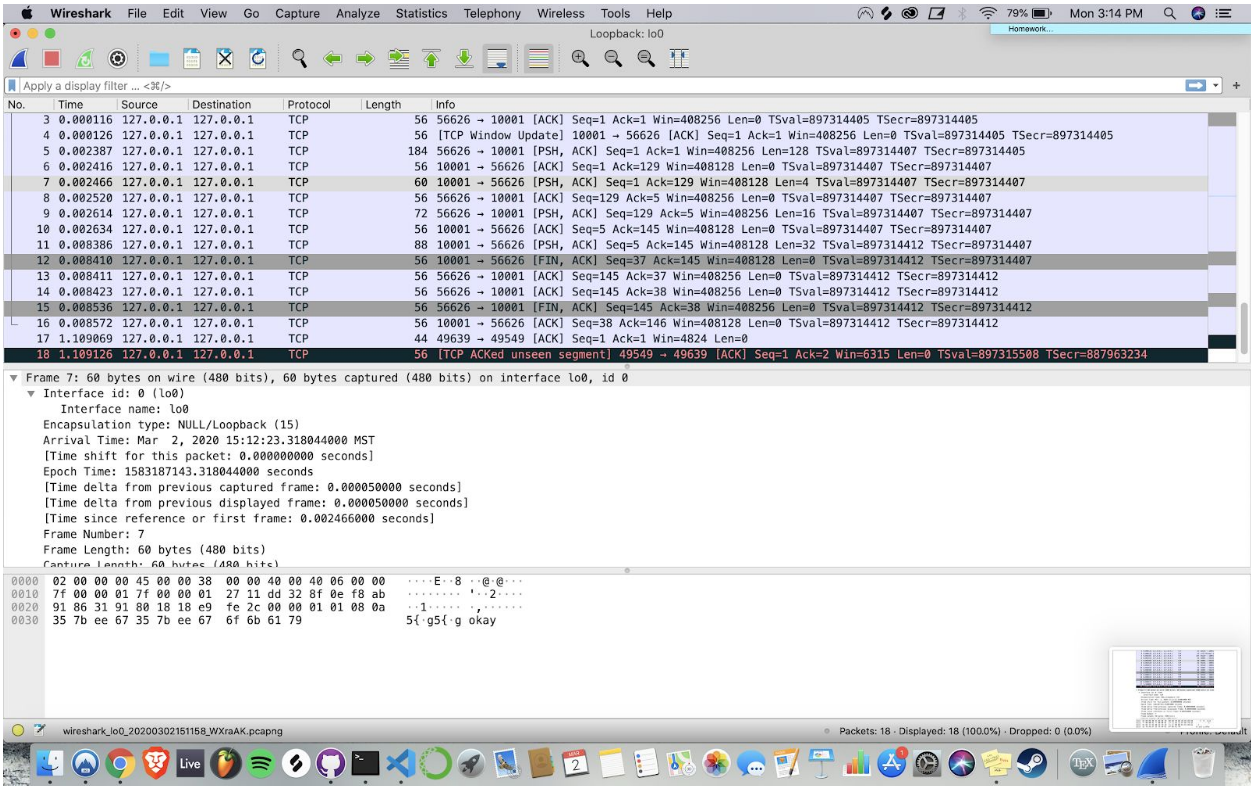
Eavesdropper is a network layer attack focusing on capturing small packets from the network transmitted by computers and searching the content for information. If a network service was lacking encryption, it would be likely for the eavesdropping attack to occur. However, in the program we built, all of our messages were encrypted by various methods. For example, in the beginning, we generated random values for salting. Then we generated the private key using RSA encryption. Those methods will prevent other parties from being able to decode or eavesdrop what was sent.

Replay Attacks

The program is secure from replay attacks because both the sender and receiver have a completely random established session key. Since it is encrypted, the keys cannot be decoded at the end of the transmission. Additionally, there is a function which verifies that the user has been authenticated before encrypting a response to the client.

Project Reflection

The crucial part of this project is that we got hands-on experience with communication between a basic client and a server. We ended up doing a numerous amount of research on the methods and libraries we used. For example, we all did our individual research on hashing algorithms and compared and contrasted pro's and con's of each. We used resources from Stackoverflow and other websites to get a clear picture of how each of the algorithms work and the percs behind them. Collectively, once we figured out a plan of attack on our implementation of the coding parts, it became easier to understand and visualize the project. The biggest takeaway we had on this project was that, when we wanted to lay our hands on a cybersecurity-related work, it's pivotal to contemplate the concepts (e.g RSA, SHA-2, AES) first before coding and execution.


Wireshark File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
79% Mon 3:14 PM
Homework...
Loopback: lo0
Apply a display filter ... <⌘/>

```
No.  Time     Source    Destination Protocol Length Info
   3 0.000116 127.0.0.1 127.0.0.1   TCP       56 56626 → 10001 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=897314405 TSecr=897314405
   4 0.000126 127.0.0.1 127.0.0.1   TCP       56 [TCP Window Update] 10001 → 56626 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=897314405 TSecr=897314405
   5 0.002387 127.0.0.1 127.0.0.1   TCP      184 56626 → 10001 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=128 TSval=897314407 TSecr=897314405
   6 0.002416 127.0.0.1 127.0.0.1   TCP       56 10001 → 56626 [ACK] Seq=1 Ack=129 Win=408128 Len=0 TSval=897314407 TSecr=897314407
   7 0.002466 127.0.0.1 127.0.0.1   TCP       60 10001 → 56626 [PSH, ACK] Seq=1 Ack=129 Win=408128 Len=4 TSval=897314407 TSecr=897314407
   8 0.002520 127.0.0.1 127.0.0.1   TCP       56 56626 → 10001 [ACK] Seq=129 Ack=5 Win=408256 Len=0 TSval=897314407 TSecr=897314407
   9 0.002614 127.0.0.1 127.0.0.1   TCP       72 56626 → 10001 [PSH, ACK] Seq=129 Ack=5 Win=408256 Len=16 TSval=897314407 TSecr=897314407
  10 0.002634 127.0.0.1 127.0.0.1   TCP       56 10001 → 56626 [ACK] Seq=5 Ack=145 Win=408128 Len=0 TSval=897314407 TSecr=897314407
  11 0.008386 127.0.0.1 127.0.0.1   TCP       88 10001 → 56626 [PSH, ACK] Seq=5 Ack=145 Win=408128 Len=32 TSval=897314412 TSecr=897314407
  12 0.008410 127.0.0.1 127.0.0.1   TCP       56 10001 → 56626 [FIN, ACK] Seq=37 Ack=145 Win=408128 Len=0 TSval=897314412 TSecr=897314407
  13 0.008411 127.0.0.1 127.0.0.1   TCP       56 56626 → 10001 [ACK] Seq=145 Ack=37 Win=408256 Len=0 TSval=897314412 TSecr=897314412
  14 0.008423 127.0.0.1 127.0.0.1   TCP       56 56626 → 10001 [ACK] Seq=145 Ack=38 Win=408256 Len=0 TSval=897314412 TSecr=897314412
  15 0.008536 127.0.0.1 127.0.0.1   TCP       56 56626 → 10001 [FIN, ACK] Seq=145 Ack=38 Win=408256 Len=0 TSval=897314412 TSecr=897314412
  16 0.008572 127.0.0.1 127.0.0.1   TCP       56 10001 → 56626 [ACK] Seq=38 Ack=146 Win=408128 Len=0 TSval=897314412 TSecr=897314412
  17 1.109069 127.0.0.1 127.0.0.1   TCP       44 49639 → 49549 [ACK] Seq=1 Ack=1 Win=4824 Len=0
  18 1.109126 127.0.0.1 127.0.0.1   TCP       56 [TCP ACKed unseen segment] 49549 → 49639 [ACK] Seq=1 Ack=2 Win=6315 Len=0 TSval=897315508 TSecr=887963234
```

```
▼ Frame 7: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface lo0, id 0
  ▼ Interface id: 0 (lo0)
       Interface name: lo0
    Encapsulation type: NULL/Loopback (15)
    Arrival Time: Mar  2, 2020 15:12:23.318044000 MST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1583187143.318044000 seconds
    [Time delta from previous captured frame: 0.000050000 seconds]
    [Time delta from previous displayed frame: 0.000050000 seconds]
    [Time since reference or first frame: 0.002466000 seconds]
    Frame Number: 7
    Frame Length: 60 bytes (480 bits)
    Capture Length: 60 bytes (480 bits)
```

```
0000  02 00 00 00 45 00 00 38  00 00 40 00 40 06 00 00   ····E··8 ··@·@···
0010  7f 00 00 01 7f 00 00 01  27 11 dd 32 8f 0e f8 ab   ········ '··2····
0020  91 86 31 91 80 18 18 e9  fe 2c 00 00 01 01 08 0a   ··1····· ·,······
0030  35 7b ee 67 35 7b ee 67  6f 6b 61 79               5{·g5{·g okay
```

wireshark_lo0_20200302151158_WXraAK.pcapng   Packets: 18 · Displayed: 18 (100.0%) · Dropped: 0 (0.0%)   Profile: Default